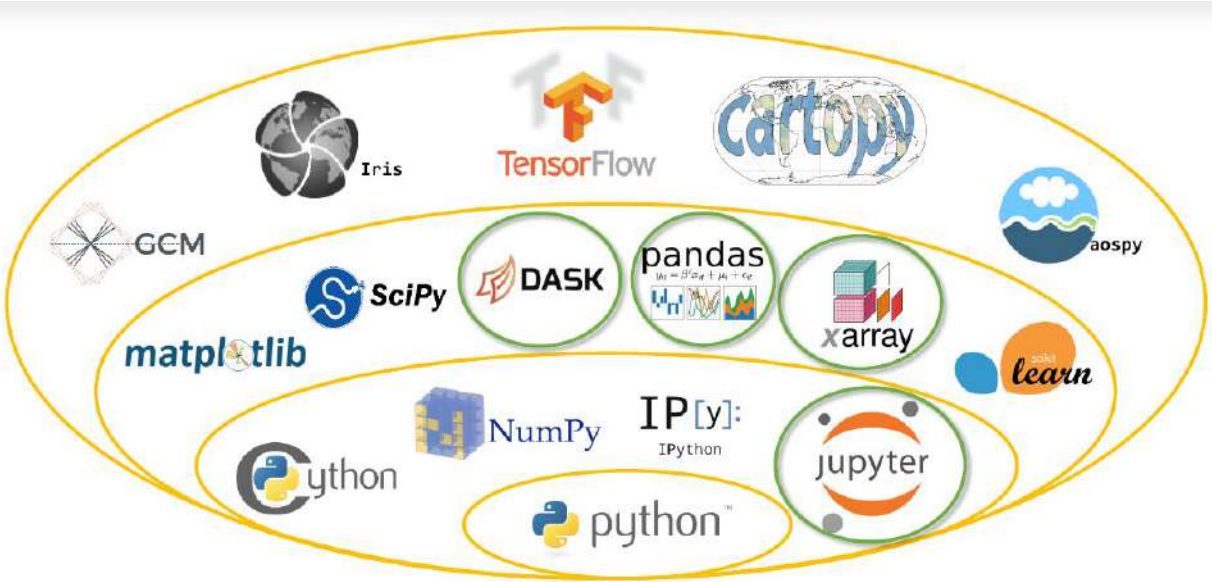
VRE et Accès aux Données

Atelier ODATIS, 10-11 Oct 2024

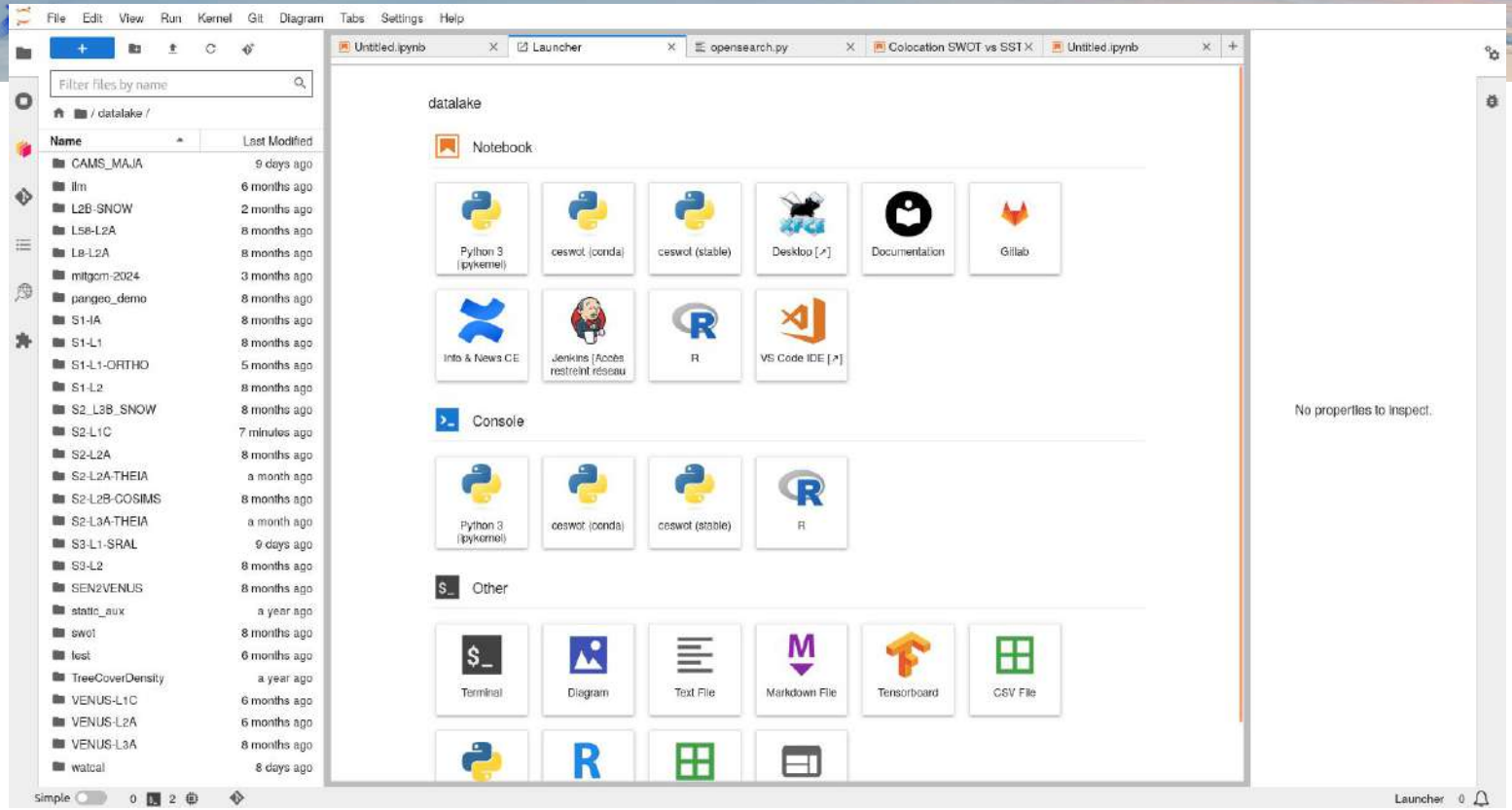
A (Pangeo) Virtual Research Environment

https://pangeo.io



Data + Infrastructure + Software + Community

VRE example (CNES)



Expertise Center documentation

Search

Home Quick Start User Guide Data Access Tutorial & How-To Resources

Microwave Expertise Center

The Microwave Expertise Center (MWEC) is a supercharged JupyterLab web based prototyping environment.
It provdes a set of tools dedicated to developping and running science algorithms, geospatial information oriented.

Getting started

Data access

How to access satellite altimetry data from various data catalogs?

Tutorials

A gallery filled with tutorials of data exploration and analysis grouped by thematic surface

Resources

Useful documentation links and contact info

Jupyter

edit and run code locally or remotely (jupyterhub) within a web navigator

write **notebooks**

mix in text, processing code, result plots : full analysis, story, course/tutorial, workflows…

works with Python, Julia, R, …

shareable (git,..), repeatable > **open science** - numerous recipes, science work, tutorials,...

available on many infrastructures, science clouds,...

many complementary packages

Streamlining scientific analysis workflows

- workflow editing and execution environment: **Jupyter / JupyterLab**
- remote data access and processing: **JupyterHub**
- data selection and loading : **Intake**, **STAC**, **Opensearch**
- abstraction over N-dimensional data arrays: **Xarray**, Pandas
- performances:
  - cloud optimized format: **kerchunk**
  - distributed processing: **Dask**
- all useable within **Jupyter notebooks**

Jupyter, Xarray and **Dask** now widely used and popular in scientific community (Ocean, Weather, Climate)

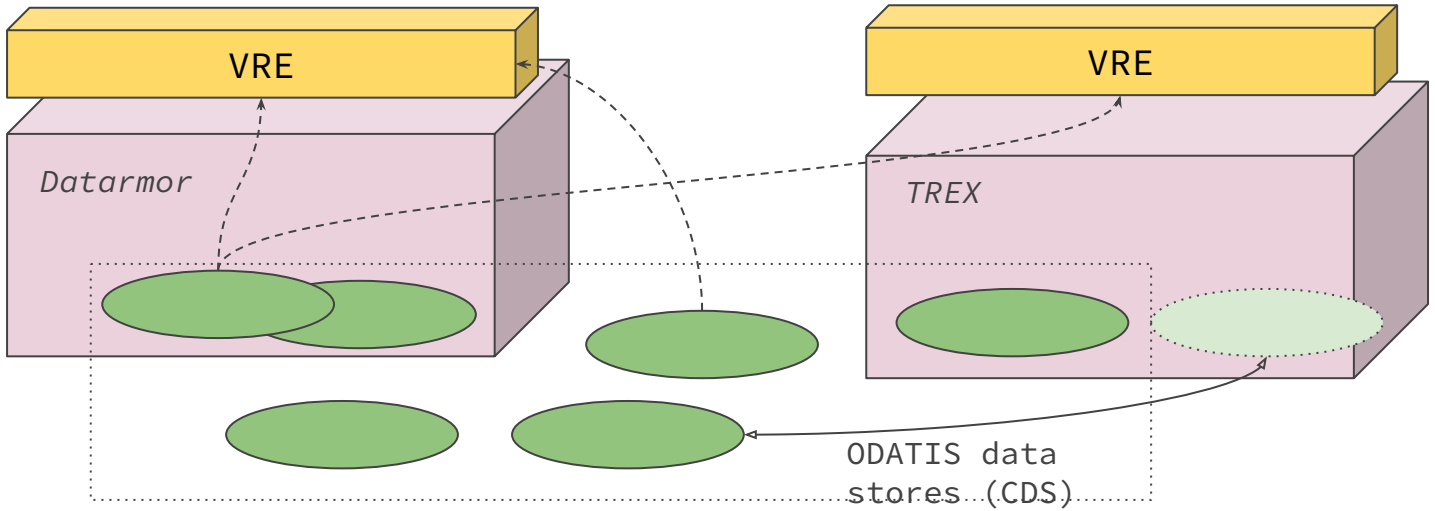
ODATIS “Datalake”

interoperable data access protocols

mirror (user triggered or sustained by infra)

user applications

infrastructures (Data Terra)

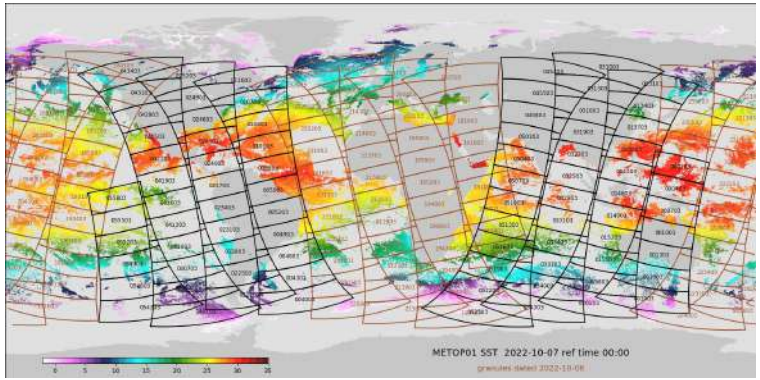


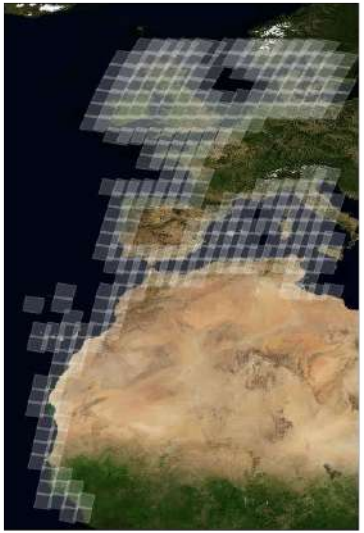
A typical VRE user workflow

- Opening a Jupyter session
- Searching (Sextant) data catalogue or thematic catalogue
- from product landing page:
  - **get data access points** (posix, S3, OpenDAP,...)
  - **get data search end-points** (STAC)
  - (optional) get sample notebook on dataset usage
- starting to write my own notebook
  - search assets to open matching user criteria
  - read data
  - do some stuff…
- more advanced: from manual analysis to automation/repetition : batch processing, workflow managers, … => requires data **persistency**, automatic **updates**, …

searching for geospatial data

typical of low orbit satellite data, performing multiple revolutions per day - or buoy/ship => selecting granules (swath sections, profiles,... - files) matching a given time frame and area of interest





1 Easy Earth Observation Data Selection with STAC API

Users of spatial temporal data are often burdened with building unique pipelines for each different collection of data they consume. The STAC community has defined a specification to remove this complexity and spur common tooling. In particular it offers to users an harmonized API to browse catalogues of Earth Observation data collections and select data of interest within these collection.

This notebook demonstrates the basic usage of the STAC API for Ifremer's catalogue of Earth Observation (EO) data. It goes through the main functions useful to query data over situations of interest.

The URL for Ifremer STAC API is: http://visi-common-docker1.ifremer.fr:8080

To run the following handbook, you recommend you use the predefined `STAC` conda environment, or create your own conda environement, in which case you will need to install a minimum list of packages as follow:

- `pystac_client` for querying the STAC API (see the full documentation of this python client library at: https://pystac-client.readthedocs.io)
- `folium` for map visualisation

1.1 Browsing the STAC catalogue

The first step is to open the catalogue serverd by Ifremer STAC API:

[1]:
```
from pystac_client import Client

#catalogue = Client.open("http://visi-common-docker1.ifremer.fr:8080")
catalogue = Client.open("https://stac-api.ifremer.fr/")
catalogue
```

[1]:
```
<Client id=stac-ifremer>
```

It is possible to display the list of served EO datasets by iterating over the catalogue *collections*

[2]:
```
for item in catalogue.get_collections():
    print(item)
```

```
<CollectionClient id=argo_cycles>
<CollectionClient id=ascat_a_l2b_12km_knmi>
<CollectionClient id=ascat_b_l2b_12km_knmi>
<CollectionClient id=avhrr_sst_metop_a-osisaf-l2p-v1.0>
<CollectionClient id=avhrr_sst_metop_b-osisaf-l2p-v1.0>
<CollectionClient id=avhrr_sst_metop_c-osisaf-l2p-v1.0>
<CollectionClient id=cfosat_sca_l2a___>
<CollectionClient id=iwwoc_swi_l2s___>
<CollectionClient id=cfosat_swi_l2anad>
<CollectionClient id=cryosat2_igdr>
<CollectionClient id=gpm_l2_ku_v6a>
<CollectionClient id=hy2b_l2a___>
<CollectionClient id=jason2_igdr>
<CollectionClient id=jason3_igdr>
<CollectionClient id=s1a_wv_ocn>
<CollectionClient id=s1b_wv_ocn>
<CollectionClient id=s3a_sr_2_wat>
<CollectionClient id=s3b_sr_2_wat>
<CollectionClient id=scatsat1_l2b_knmi>
<CollectionClient id=cciseastate_l2p_alt_jason_1>
<CollectionClient id=cciseastate_l2p_alt_jason_2>
<CollectionClient id=cciseastate_l2p_alt_jason_3>
<CollectionClient id=cciseastate_l2p_alt_envisat>
<CollectionClient id=cciseastate_l2p_alt_cryosat_2>
<CollectionClient id=cciseastate_l2p_alt_sentinel_3a>
<CollectionClient id=cciseastate_l2p_alt_saral>
<CollectionClient id=avhrr_sst_metop_a_glb-osisaf-l3c-v1.0>
<CollectionClient id=avhrr_sst_metop_b_glb-osisaf-l3c-v1.0>
<CollectionClient id=goes16-osisaf-l3c-v1.0>
<CollectionClient id=seviri_io_sst-osisaf-l3c-v1.0>
<CollectionClient id=seviri_sst-osisaf-l3c-v1.0>
```

To get a more detailed description of a given collection, use `get_collection` method with one of the identifiers of above list of collections.

[3]:
```
collection = catalogue.get_collection('avhrr_sst_metop_b-osisaf-l2p-v1.0')
collection
```

1.2 Data selection

We will now search data from the above collection over an area (defined in `bbox`) and period of interest (defined in `datetime`). The `bbox` is defined as a list [lon min, lat min, lon max, lat max]. Note also the format for expressing a date/time interval.

[4]:
```
search = catalogue.search(
    #max_items=10,
    #collections=['avhrr_sst_metop_a-osisaf-l2p-v1.0', 'argo_cycles'],
    datetime='2023-01-01T00:00:00Z/2023-01-31T23:59:59Z',
    collections=['avhrr_sst_metop_b-osisaf-l2p-v1.0'],
    bbox=[-10, 40, 5, 50],
)
```

Note that the `matched` method of `pystac-client` that returns the number of found results does not work with Ifremer server API (it is expected from some APIs):

[5]:
```
print(f"{search.matched()} items found")
```

```
291 items found
```

[6]:
```
items = search.item_collection()
print(f"{len(items)} items found")

i = 0
for item in items:
    print(item)
    i += 1

print(i)
```

```
10010 items found
<Item id=avhrr_sst_metop_b-osisaf-l2p-v1.0:20230101084003-OSISAF-L2P_GHRSST-SSTsubskin-AVHRR_SST_METOP_B-sstmgr_metop01_20230101_084003-v02.0-fv01.0.nc>
<Item id=avhrr_sst_metop_b-osisaf-l2p-v1.0:20230102100103-OSISAF-L2P_GHRSST-SSTsubskin-AVHRR_SST_METOP_B-sstmgr_metop01_20230102_100103-v02.0-fv01.0.nc>
<Item id=avhrr_sst_metop_b-osisaf-l2p-v1.0:20230102194603-OSISAF-L2P_GHRSST-SSTsubskin-AVHRR_SST_METOP_B-sstmgr_metop01_20230102_194603-v02.0-fv01.0.nc>
<Item id=avhrr_sst_metop_b-osisaf-l2p-v1.0:20230103093703-OSISAF-L2P_GHRSST-SSTsubskin-AVHRR_SST_METOP_B-sstmgr_metop01_20230103_093703-v02.0-fv01.0.nc>
<Item id=avhrr_sst_metop_b-osisaf-l2p-v1.0:20230103224603-OSISAF-L2P_GHRSST-SSTsubskin-AVHRR_SST_METOP_B-sstmgr_metop01_20230103_224603-v02.0-fv01.0.nc>
<Item id=avhrr_sst_metop_b-osisaf-l2p-v1.0:20230104204003-OSISAF-L2P_GHRSST-SSTsubskin-AVHRR_SST_METOP_B-sstmgr_metop01_20230104_204003-v02.0-fv01.0.nc>
<Item id=avhrr_sst_metop_b-osisaf-l2p-v1.0:20230105202503-OSISAF-L2P_GHRSST-SSTsubskin-AVHRR_SST_METOP_B-sstmgr_metop01_20230105_202503-v02.0-fv01.0.nc>
<Item id=avhrr_sst_metop_b-osisaf-l2p-v1.0:20230106083703-OSISAF-L2P_GHRSST-SSTsubskin-AVHRR_SST_METOP_B-sstmgr_metop01_20230106_083703-v02.0-fv01.0.nc>
<Item id=avhrr_sst_metop_b-osisaf-l2p-v1.0:20230106115503-OSISAF-L2P_GHRSST-
```

Xarray

Enrich and make more flexible the usage of numerical arrays, build relations between these arrays

A Python package providing 2 main classes:

- **DataArray**: a labelled multidimensional variable and its coordinates ~ enriched numpy array
- **Dataset**: group several variables sharing possibly coordinates ~ NetCDF file content

DataArray

- **values** : data of the variable
- **dims** : dimensions of each array axis (for instance: ('x', 'y', 'z'), (‘time’, ‘lat’, ‘lon’),...)
- **coords** : associated arrays (coordinates), to use for indexing along each dimension ~ tick labels
- **attrs** : metadata (attributes) of the variable as a dictionary (for instance: units, standard_name, comment,...)

DataSet

A Dataset contains several DataArrays indexed by a variable name

Xarray

Xarray operations:

- labelled selection and operations (on named axes)
- piping operations
- ideal for work with gridded time series (models, L3/L4)
  - statistical operations on labelled axes
  - groupby
  - resampling
- blends with plotting frameworks (matplotlib, cartopy)

Also checkout pandas (https://pandas.pydata.org), for another data model more oriented toward Data Frames and Time Series

```
ds_all = xr.open_mfdataset('/work/ALT/odatis/briolf/data/sst/NOAA_NCDC_ERSST_v3b_SST-*.nc', combine='by_coords')
ds_all
```

```
sst_clim = ds_all.sst.groupby('time.month').mean(dim='time')
sst_clim
```

```
# Anomalies de SST
sst_anom = ds_all.sst.groupby('time.month') - sst_clim

# Calcul de l'index ENSO
sst_anom_nino34 = sst_anom.sel(lat=slice(-5, 5), lon=slice(190, 240))
sst_anom_nino34
```

```
sst_anom_nino34_mean = sst_anom_nino34.mean(dim=('lon', 'lat'))
oni = sst_anom_nino34_mean.rolling(time=3).mean()
```

From persistence to memory

Data formats are **heterogeneous**

Some reading libraries provide the mapping from native format to numerical structures, e.g xarray (NetCDF, Zarr, grib, HDF5,...), cerbere, …

Some access protocols provide also this abstraction (by handling upstream the format issue):

- OpenDAP -> integrates well with several API (Xarray, C/C++,...)
- OGC protocols

Benefit of these protocols in addition to remote access and subsetting

Some cataloguing solutions include readers for seamless in memory loading:

- intake

Intake

A package for finding, investigating, loading and disseminating data

A Catalogue of data collections

associates file collection, default reader, default plotting : series of files can be loaded into memory as a multidimensional array

=> **provides abstraction over data location and data format**

works well with regularly organized data (gridded data), limitations with scattered data (satellite, in situ,...)

example: https://gitlab.ifremer.fr/cersat/recipes/-/blob/main/intake/lops_intake.ipynb

```
plugins:
    source:
        - module: intake_xarray
sources:

    ANDRO:
        description: An Argo-based deep displacement dataset (http://doi.org/10.17882/47077)
        args:
            urlpath: '/home5/pharos/REFERENCE_DATA/ANDRO/DATA/
androPTS_ANDRO_all_2017_dep_20180416T153202_final.dat'
          csv_kwargs:
            delim_whitespace: True
            header:
            names: ['longitude deep','latitude deep','Reference parking pressure','Parking temperature','Parking
salinity','Julian days deep','U deep','V deep','Error U deep','Error V deep','longitude surf 1','latitude surf
1','Julian days surf 1','U surf 1','V surf 1','Error U surf 1','Error V surf 1','longitude surf 2','latitude surf
2','Julian days surf 2','U surf 2','V surf 2','Error U surf 2','Error V surf
2','24','25','26','27','28','29','30','31','32','33','WMO','Cycle number','Profile number']
        driver: csv

    AVISO_DT:
        description: Global altimetry fields from 1993 up to 2018
        driver: netcdf
        args:
            urlpath: '/home5/pharos/REFERENCE_DATA/ALTIMETRY/DATA/REP/dataset-duacs-rep-global-merged-allsat-phy-l4-
v3/????/dt_global_allsat_phy_l4_*.nc'
            concat_dim: time
            chunks: {'time':10000}

    AVISO_NRT:
        description: Global altimetry fields in Near Real Time, from 2017 up to now
        driver: netcdf
        args:
            urlpath: '/home5/pharos/REFERENCE_DATA/ALTIMETRY/DATA/NRT/dataset-duacs-nrt-global-merged-allsat-phy-
l4/????/??/nrt_global_allsat_phy_l4_*.nc'
            concat_dim: time
            chunks: {'time':10000}

    CGLORS-temp:
        description: CGLORS temperature reanalysis (1/4 deg lat x 1/4 deg lon, 50 vertical levels, from 1982 to 2013)
        driver: netcdf
        args:
            urlpath: '/home5/pharos/REFERENCE_DATA/OCEAN_REP/CGLORS/dataset-global-reanalysis-phys-001-011-ran-it-
cglors-monthly-t-ssh/*.nc'
            concat_dim: time
```

Conclusion

many (mainly python) packages and tools now available to speed up science workflows

- relieve users from burden such as data selection, access, transfer and reading
- take advantage of large processing resources concentrated within data centers - cloud or HPC
- sharing - collaborative work ; users at different places can see and execute the same thing

Importance dans ODATIS de mettre en place les couches basses : accès aux données - protocoles standardisés et homogènes dans ses différentes composantes